Damien Chiem Exercise 3

MAT 115

Exercise #3

Section 2.6 is about the basic object in R, which is a vector. The columns of a dataframe are vectors, but a vector can be an object by itself.

You can build a vector by entering its values yourself. There are plenty of examples in the book. Here is another one. The first exam scores (out of 50) in one of my Biostatistics courses some years ago are the following numbers: *48,47,39,46,45,50,50,41,50,47,41,50,47,47,47,42,47,46,46*. You make a vector called `exam1` out of these numbers like so (note the use of the `c` in front):

```
exam1 <- c(48,47,39,46,45,50,50,41,50,47,41,50,47,47,47,42,47,46,46)
exam1
```

```
##  [1] 48 47 39 46 45 50 50 41 50 47 41 50 47 47 47 42 47 46 46
```

Explore the help documentation about the `c` function.

The scores are not listed alphabetically, or in descending order, or anything like that. They are ordered randomly.

Now enter the following set of numbers into a vector called `exam2`:
48,46,42,47,45,50,45,43,46,46,39,50,47,45,31,31,49,45,39

```
# Enter the second exam scores here. You can get rid of this comment.
#help('c')
exam2 <- c(48,46,42,47,45,50,45,43,46,46,39,50,47,45,31,31,49,45,39)
exam2
```

```
##  [1] 48 46 42 47 45 50 45 43 46 46 39 50 47 45 31 31 49 45 39
```

These are, naturally, the second exam scores for the same course. The scores are listed in the same order as the first exam scores. So the first student on the list got 48 on the first exam and also 48 on the second exam.

You can also build a new vector from other vectors. For example, suppose we want to figure out if students generally do better on exam 2 compared to exam 1. We can make a new vector called `improve` that is the difference between scores on exam 2 and scores on exam 1, then take a look at it:

```
improve <- exam2-exam1
improve
```

```
##  [1]   0  -1   3   1   0   0  -5   2  -4  -1  -2   0   0  -2 -16 -11   2  -1  -7
```

What do you think? Does it seem like students do better on exam 2 than on exam 1? What other tool (or tools) would help you determine this? It seems that students did not do better on exam 2. a lot of the differences are 0 or Negative as opposed to a positive difference. You can also find the mean values between the two and compare them to see which exam had a better score.

We can make a dataframe out of these vectors by using the `data.frame` command. But before we do that, we should assign ID numbers to the students so that we can identify them easily. We might as well use integers from 1 to however many students there are. The book mentions there are (at least) two ways to do this: we can use the `seq` command, or we can use the shortcut `1:19`. Let's use the `seq` command:

```
ID <- seq(1,length(exam1))
ID
```

```
##  [1]  1  2  3  4  5  6  7  8  9 10 11 12 13 14 15 16 17 18 19
```

What does the `length` command do? Answer here. Why is this a better option than the 1:19 shortcut? length('[vector]') gives you the length of a given vector in this context. The reason this is better than 1:19 is because if you change the vector size, length will automatically recalculate to the updated value. This way, you do not need to constantly update it manually and it allows it work with many different vectors.

Explore the `seq` help documentation. What does the by argument in this function do? Explain it in your own words. Also, create some code that successfully uses the by argument.

```
# Write code here.
help(seq)
```

```
## starting httpd help server ... done
```

```
#The "by" argument allows the sequence to increase by a certain amount. If by = 2, then the numbers wil
seq(0, 20, 2)
```

```
##  [1]  0  2  4  6  8 10 12 14 16 18 20
```

Note: we can also construct the `ID` vector by using the following code:

```
ID <- 1:length(exam1)
ID
```

```
##  [1]  1  2  3  4  5  6  7  8  9 10 11 12 13 14 15 16 17 18 19
```

As the book mentions, there are many ways to accomplish the same task in R. Maybe too many.

Now let's construct the dataframe:

```
exams <- data.frame(ID, exam1, exam2,improve)
exams
```

```
##    ID exam1 exam2 improve
## 1   1    48    48       0
## 2   2    47    46      -1
## 3   3    39    42       3
## 4   4    46    47       1
## 5   5    45    45       0
## 6   6    50    50       0
## 7   7    50    45      -5
## 8   8    41    43       2
## 9   9    50    46      -4
## 10 10    47    46      -1
## 11 11    41    39      -2
## 12 12    50    50       0
## 13 13    47    47       0
## 14 14    47    45      -2
## 15 15    47    31     -16
## 16 16    42    31     -11
## 17 17    47    49       2
## 18 18    46    45      -1
## 19 19    46    39      -7
```

Suppose we want to take a look at student #10. Since our dataframe is small, we can just inspect the spreadsheet, but for large datasets this might not be practical. We can just extract the data for student #10 by using the indexing command for matrices. Do you remember how to do this?

```
# Write your code here.
exams[10,]
```

```
##    ID exam1 exam2 improve
## 10 10    47    46      -1
```

We can also do this by looking at the vectors we have created:

```
c(exam1[10],exam2[10],improve[10])
```

```
## [1] 47 46 -1
```

Now try to extract the data for student #1 and student #9 in order to compare them. Try to do it in one line of code.

```
# Write your code here.
c(exams[1,], exams[9,] )
```

```
## $ID
## [1] 1
##
## $exam1
## [1] 48
##
## $exam2
## [1] 48
##
```

```
## $improve
## [1] 0
##
## $ID
## [1] 9
##
## $exam1
## [1] 50
##
## $exam2
## [1] 46
##
## $improve
## [1] -4
```

Section 2.7 is about `coercion` where we (or R) change the type of variables in a dataframe or a vector. As the book says, this is a large source of confusion and frustation when using R. For example, suppose you want to add quiz scores to the dataframe we built above. But for some reason, the number 10 is entered as a character "10".

```
quiz <- c(7,7,9,7,9,10,10,10,7,10,7,"1o",8,7,10,9,7,8,9)
```

R does not complain! It silently coerced the quiz scores into characters. When you try to calculate the average quiz score, this is what happens:

```
mean(quiz)
```

```
## Warning in mean.default(quiz): argument is not numeric or logical: returning NA
```

```
## [1] NA
```

Why do you think the default coercion is to a character vector and not numeric? The character data type might be more prioritized in the hierarchy of data types in R.

You can't find the average of a bunch of characters. And R thinks that the quiz scores are characters.

```
class(quiz)
```

```
## [1] "character"
```

You can force R to treat the entries in `quiz` as integers:

```
mean(as.integer(quiz))
```

```
## Warning in mean(as.integer(quiz)): NAs introduced by coercion
```

```
## [1] NA
```

What happens if you mistakenly enter `10` as `1o` (i.e., using the letter `o` instead of the digit `0`)? Try it out! (See also section 2.7.1.)

Write your answer here: It throws an error that says: NAs introduced by coercion

One more thing: in the dataframe `exams`, the `ID` variable consists of integers.

```
class(exams$ID)
```

```
## [1] "integer"
```

But `ID` really should be characters, since they are used only to identify students. It doesn't make sense to calculate the average ID number, for example. So we really should change the class of `ID` to `character`.

```
exams$ID <- as.character(exams$ID)
```

Note that you can overwrite an old vector (in a dataframe or not) by a new vector. R doesn't complain that the new name is the same as the old name.

Check that the variable `ID` is now classified as `character`.

```
class(exams$ID)
```

```
## [1] "character"
```

One final thing. You might want to save the dataframe you have been working on. One option is to use the `save()` command.

```
save(exams, file="exams.rda")
```

You should look at your working directory (folder) and check that a file called "exams.rda" is now in there. The directory should be the same directory that contain your RMarkdown file. The ".rda" extension identifies the file as an R dataframe.

One thing to be aware about `save()` is that it saves everything about the dataframe, *including its name*. So you might as well give it the same name as the original. If you give it a different name, say `savedexams`, then when you load it again in a different R session, you might think that the dataframe is called `savedexams`, but it isn't. It's still called `exams`.

To load the saved dataframe into another R session, you use the `load()` command, of course. Don't forget to use double quotes around the name of the saved dataframe.

```
# Try doing exactly that. Follow these instructions.
# Quit this R session and begin a new one.
# Replace the comment with the appropriate `load()` command.
# See if the `exams` dataframe works normally.
save(exams, file="savedexams.rda")
```

If you are interested in even more practice, work through the exercises in section 2.8 of the textbook: rafalab.dfci.harvard.edu/dsbook/r-basics.html#exercises-2.